BEST AVAILABLE IMAGES

Defective images within this document are accurate representations of the original documents submitted by the applicant.

Defects in the images may include (but are not limited to):

- BLACK BORDERS
- TEXT CUT OFF AT TOP, BOTTOM OR SIDES
- FADED TEXT
- ILLEGIBLE TEXT
- SKEWED/SLANTED IMAGES
- COLORED PHOTOS
- BLACK OR VERY BLACK AND WHITE DARK PHOTOS
- GRAY SCALE DOCUMENTS

IMAGES ARE BEST AVAILABLE COPY.

As rescanning documents *will not* correct images, please do not report the images to the Image Problem Mailbox.

(12) INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
1 March 2001 (01.03.2001)

PCT

(10) International Publication Number
WO 01/14961 A2

(51) International Patent Classification[7]: G06F 9/00

(21) International Application Number: PCT/US00/24336

(22) International Filing Date: 28 August 2000 (28.08.2000)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:
60/150,766 26 August 1999 (26.08.1999) US
60/210,334 12 June 2000 (12.06.2000) US

(71) Applicant: PARABON COMPUTATION [US/US]; Suite 1000, 3930 Walnut Street, Fairfax, VA 22030 (US).

(72) Inventors: ARMENTROUT, Steven, L.; 11879 St. Trinians Ct., Reston. VA 20191 (US). O'CONNOR, James; 11625 Ayreshire Road, Oakton, VA 22124 (US). GANNON, James; 3817 Inverness Road, Fairfax, VA 22030 (US). SLETTEN, Brian; Apt. 906, 900 N. Stuart Street, Arlington, VA 22203 (US). CIER, Sean; Apt. 12, 11800 Federalist Way, Fairfax, VA 22030 (US). CARLSON, Sarah; 13236 Memory Lane, Fairfax, VA 22033 (US). DAVIS, Jonathan; Apt. 312, 202 Columbia Road NW, Washington, DC 20009 (US). DUPERTUIS, Greg; 7888 Leroux Lane, Manassas, VA 20112 (US). MCCLOUGHLIN, Scott; 2827 28th Street, #14, Washington, DC 20001 (US). DAVIES, Antony; 11733 Stuart Mill Road, Oakton, VA 22124 (US).

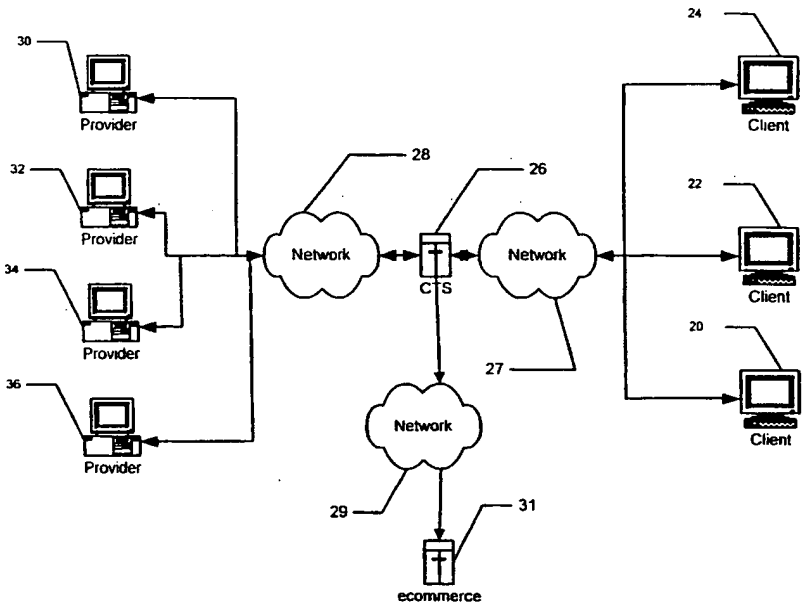
(74) Agents: ROBERTS, Jon, L. et al.; Roberts Abokhair & Mardula, LLC, Suite 1000, 11800 Sunrise Valley Drive, Reston, VA 20191 (US).

(81) Designated States *(national)*: AE, AG, AL, AM, AT, AU, AZ, BA, BB, BG, BR, BY, BZ, CA, CH, CN, CR, CU, CZ, DE, DK, DM, DZ, EE, ES, FI, GB, GD, GE, GH, GM, HR, HU, ID, IL, IN, IS, JP, KE, KG, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, MA, MD, MG, MK, MN, MW, MX, MZ, NO, NZ, PL, PT, RO, RU, SD, SE, SG, SI, SK, SL, TJ, TM, TR, TT, TZ, UA, UG, UZ, VN, YU, ZA, ZW.

(84) Designated States *(regional)*: ARIPO patent (GH, GM, KE, LS, MW, MZ, SD, SL, SZ, TZ, UG, ZW), Eurasian patent (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European

[Continued on next page]

(54) Title: SYSTEM AND METHOD FOR THE ESTABLISHMENT AND UTILIZATION OF NETWORKED IDLE COMPUTATIONAL PROCESSING POWER



(57) Abstract: A distributed computing platform using the idle computational processing power of a plurality of provider computers is disclosed. At least one networked server collects tasks from client computers, schedules and distributes the tasks to networked provider computers, and collects and returns results to client computers. A client API forms tasks and collects results. A compute engine operates on the provider computers to communicate with the server and execute tasks using idle computational power.

WO 01/14961 A2

Title of Invention: System and Method for the Establishment and Utilization of Networked Idle Computational Processing Power

Field of the Invention:

This invention relates generally to computer use over a network to form a distributed computing platform. More specifically the present invention is a system and method for use of idle time of computers that are connected to a network by users requiring significant computing power without the need for a large scale dedicated processing unit.

Background of the Invention:

When a computer is operating, but not actively performing computations for someone, it is said to be *idle.* Because of their incredible speed, modern computers are idle most of the time, not only when they are running screen savers, but *even when they are being used.* For instance, a fast typist working at top speed in a word processor uses only a fraction of the available computational capacity of a desktop PC. Although the time between a typist's keystrokes seems immeasurably small to humans, it is an eternity when measured in computer time - an eternity filled with unproductive *idle computation.* Because of this gross underutilization, it is estimated that well over 95% of the world's computational capacity presently goes to waste.

Summary of the Invention

The present invention is drawn to organization and use of the idle processing power of general-purpose computers to form a distributed computing platform.

It is therefore an objective of the present invention to provide large amounts of computational power to users without the users having to purchase a large computer for such purposes.

It is a further objective of the present invention to harness the idle computational power of many computers and make that power available to clients on an as needed basis.

It is yet another objective of the present invention to allow clients to create a virtual cluster of machines of client-definable computational power to run client tasks.

It is thus an overall goal of the present invention to utilize this heretofore underutilized computer power in a novel way as a distributed computing platform to meet the needs of users who require vast computing power but who may not have the financial wherewithal to purchase or lease large mainframe computers or other supercomputing solutions.

Just as email service providers deliver email from one user to another, a business running the system and method of the present invention will disseminate the tasks of a client's distributed computer program to multiple providers for remote execution. As the remote providers complete tasks, the providers will deliver results back to the originating client. As part of the present invention, an intermediary server delivers *computation* from providers to clients.

The Internet infrastructure to facilitate the distributed processing of the present invention requires three components:

1) Client software applications for requesting and submitting distributed processing requests and jobs. This is accomplished through a client API that allows client jobs to be coded as independent tasks. The underlying task model, while complex, is hidden from the user.;
2) Provider software called a compute engine (CE) for managing the launch and execution of tasks delivered from the intermediary server. The CE runs unobtrusively and securely on the provider's computer. It processes tasks when the computer is idle and returns results to the server when the provider computer connects to the network which, for example, and without limitation may be the Internet; and
3) A centralized task server for exchanging tasks and results between participants. The centralized task server (CTS) comprises a three-tiered architecture as more fully set forth below and is redundant, fault tolerant and scalable as traffic and clients increase. A task scheduler in the CTS matches power requested to available provider computer resources.

In this fashion, from a client's perspective, the present invention represents a gigantic virtual computer or distributed computing platform, ideally suited for performing large-scale parallel *distributed computation*, a function that was formerly the exclusive domain of traditional "chips-and-solder" supercomputers. The present invention, however, will perform such computations at a fraction of the cost and on an as-needed basis.

The business model associated with the present invention is to create a new market for idle computation power. A business running the apparatus and method of the present invention will purchase the idle computational power of millions of *providers* (individual computer owners), and deliver that idle computational power as a single entity to *clients* (technologists seeking ultra-high-performance computation).

Presently, idle computation has no value whatsoever because the market for computation is inseparable from the market for computers. Users have no means of buying, for example, larger amounts of times on disparate computers to execute a job when the user needs it and not just when a specific computer is available.

With the present invention, however, immense value can be created by a computational intermediary in the same way that financial engineers create value by constructing derivative financial instruments. In effect, the present invention *strips* excess computational capacity from provider computers, *bundles it* into quantities suitable for large-scale parallel computations, and *delivers* it to clients on an as needed basis. Providers benefit by selling a heretofore-wasted resource. Clients benefit because they can inexpensively purchase just the quantity of computation needed, thereby avoiding the normally huge fixed costs associated with purchasing a supercomputer.

The business model of the present invention creates and intermediates the new market in idle computation.

Finally, an applications research and development group will add value by utilizing its superior knowledge of distributed processing to solve high-payoff computational problems. Such a team will spur demand in two ways: First, its purchases of idle computation will stimulate market activity; second, the success of applications and research group will demonstrate to prospective clients the power and cost-effectiveness of idle time processing.

A centralized task server (CTS) **26** intermediates the needs of client applications and the power of the provider computers **30-36** to meet the needs of clients for computational power. Clients make requests via client computers **20-24** and by virtue of a client application programming interface (Client API) that allows the client to specify the amount of computing power desired, the individual rating of computers that will execute the client's job (e.g., 2 gigaflops or GF [Flops = Floating point operations per second]), how much memory and disk space will be necessary and other parameters essential to running the client's job. The CTS receives tasks from clients and assigns them to one or more providers based the characteristics of the available providers and the job characteristics specified by the client. When providers return results, the CTS forwards these back to the client application.

Each provider computer has a resident compute engine (CE) that receives and executes the various tasks that are assigned to it by the CTS. The CE runs client tasks and manages the use of idle computation power of the provider computer. The CE returns intermediate and final results to the CTS that then forwards them back to the client application.

Communication between the client application, CTS, and CE is governed by a communication protocol unique to the present invention. At a high level of abstraction, separate message and data channels exist. The message channel is a bi-directional channel between clients and the server that is used to pass high-level messages that facilitate distributed processing. The data channel is a channel used to pass large blocks of data called data elements and executable elements. Executable elements are blocks code developed by clients or a third-party. This code represents tasks or portions of tasks. Data elements are blocks of data that are either input to or output of client tasks. High-level messages transmitted on the message channel contain references to data and executable elements. These items are separately transferred on the data channel on an as needed basis.

The details of the client development environment, CTS, CE, and protocol the protocol that these components use to communicate will be covered in subsequent sections.

Referring first to **Figure 2** the general flow of the present invention is illustrated. Clients **10**, that is, those entities desiring execution of computationally intensive tasks, launch

allows a client application to manage the execution of jobs consisting of a very large number of tasks, and a set of tools that support the monitoring and control of very large jobs. These are described in more detail below.

The runtime API defines the environment in which client task run. The runtime API consists of a set of function calls that the client task will exercise during execution to interact with its environment. This API contains calls that the task can use to:

- Obtain access to the initialization parameters for the task.
- Obtain access to data elements needed to perform its function.
- Perform a checkpoint so that if the task is interrupted, it can restart at the checkpoint and therefore reduce the loss of work.
- Send interim and final results back to the server.

In addition, the API defines entry points that allow the environment to control the task. These include:

- Start a task (note that restarting a task is a special case of starting a task).
- Stop (suspend) a task.
- Request a checkpoint.
- Request a status.
- Terminate a task.

This is not an exhaustive list of functions and is only illustrative of what can be accomplished via the runtime API.

There can be multiple implementations of this API which allow the task to run in multiple environments. The two most important implementations for the purpose of this invention are the implementation in the CE and the run local implementation included in the client development environment. The CE implementation allows the task to run in the CE on a provider's box (regardless of platform). The run local implementation allows clients to execute tasks on their local client machine. This enables debugging and testing before the job is distributed. However, the client could elect to run an entire job (or a certain number or percent of tasks) locally.

(percentage), the work done on completed tasks, and the time spent per completed task. These statistics are illustrative only. Other statistics may also be gathered as desired by the client. The information that is monitored is displayed in both graphic and text form.

These tools can also be used to support other general-purpose functions such as starting and stopping individual tasks or jobs, removing individual tasks or jobs, etc.

Referring to **Figure 3**, one embodiment of the CE architecture of the present invention is illustrated. The CE is the software unit that resides on a provider computer and accomplishes the timely and secure execution of client tasks. The CE **84** manages the interface between the provider computer and the server of the present invention, monitors the state of the provider computer's machine, schedules the execution of tasks on the provider computer, and manages all communication with the server of the present invention. For example, the CE **84** receives tasks from the CTS, posts results to the server, provides status of tasks and receives messages to remove, suspend or resume tasks.

The CE 84 can be configured to execute tasks continuously, albeit at low priority (so as not to interfere with a provider's normal use of their computer), or to start execution of tasks when a provider's computer is idle (i.e., not being interactively used) and stop execution immediately upon provider interaction with their computer. In this way, the CE is designed to be unobtrusive to the provider.

CE **84** communications with the server and execution of tasks occur independently. For example, even if a provider is actively using their computer, the CE communicates with the server, provided a network connection is available (e.g., when a provider has dialed into the Internet). In so doing, the CE receives elements that are placed in the task queue **86** to be executed whenever possible. Thus, obtaining and staging of client tasks takes place at a time independent from the execution of those tasks. Making CE communications independent from CE task execution adds to the efficiency of the overall system.

The CE also includes a dialing scheduler that can be preset by the provider to dial out to the server of the present invention at configurable intervals and only during preset periods. Thus, for example, the provider can direct the dialer to dial out to the server every ten minutes, but only between the hours of 2 a.m. to 4 a.m.

accessible to the user such that accounts can be accessed, amount of time associated with use of the users computer can be determined.

The graphic window of the CE is displayed for the provider when the CE detects the host computer is idle. Its presence indicates to the provider that the system is processing, or is ready to process tasks in the queue, and has a number of functions. The window can show the state of processing, that is whether tasks are being processed at a particular moment or not. Further, within the window are located various other buttons that may be clicked to determine the status of the provider's account. In addition, advertising or other messages may be presented in the CE window, which will allow a user to proceed to a particular website to view further information concerning the messages. Thus, the CE window can operate in the nature of a browser.

The size and placement of the CE window during "pop up" is configurable by the user. When the CE window is displayed, the system seizes control of the cursor and locates it at any configurable spot within the CE window. Thus the cursor can be placed in a spot where messages are to be displayed or in a location to actuate any functionality that is available to the provider while the CE is operating, such as the quick disappearance of the display window as described above.

The user can also confine the CE to a system tray if desired. In this case, the icon or panel in the tray can be used to indicate, among other things, engine status, the arrival of new display content or upgrade packages, or Internet connectivity status. Alternatively, the CE can be configured and executed as a "daemon" having only a command-line user interface.

The CE also supports the dynamic display of marketing content in the CE window to the provider computer. The server notifies the CE when new content is available and where it can be downloaded.

In the event that the user desires to use the computer for purposes other than to run elements of the present invention, the user will simply type a key or cause the cursor to be moved outside the CE window. As soon as this occurs, the CE will shut down all of the processing of the present invention in an orderly fashion. Alternatively, the CE can be configured to execute tasks constantly, albeit at low priority, in which case the CE window

The runtime then requests the data elements for the task **106** over the runtime to CE channel, and the CE returns them in response over the CE to runtime channel.

The runtime then establishes a task context that the task uses to communicate with the runtime system **108**.

The runtime establishes and begins the client task **110**. The executable elements for this task having been provided to the runtime via the class path provided for the JVM when the JVM was started. The client task then runs in a separate protection domain that prevents it from accessing the network or the disk. This protection domain is established through the permissions granted to the task in the task policy file.

The client task interacts with the runtime through the task context object established for it by the runtime **112**. The runtime calls run with the client tasks privileges when possible. If the runtime needs to perform an operation requiring additional privileges, i.e., accessing a particular data element, the runtime executes a privilege block that takes on the minimum privilege for the minimum amount of time required to complete the operation.

The client interacts with the runtime to get task parameters and to access task elements **114**. As an optional step, when the client task can send out temporary results using the set status call to the runtime **116**, these results are passed to the CE over the runtime to CE channel. The CE then sends these interim results in the form of a task status to the server.

As an additional option, when the client task can create a task checkpoint using the checkpoint call to the runtime **118**, the checkpoint is passed to the CE over the runtime to CE channel. The CE serializes the checkpoint to disk so it is available in case a task restart is required.

When finished, the client task does a set status runtime call **120** with the final results and exits. The results are passed through the CE over the runtime to CE channel. The CE then sends these results in the form of a task status message to the server. After the CE sends a final task status noting that the task is complete **120**, the space that is consumed by the task definition and related data can be reclaimed for use in running other tasks.

Once the task completes **122**, the runtime closes down its connections and exits.

One of the most critical functions of the CTS is task scheduling. Part of the scheduling process is to provide clients of the present invention, who desire to obtain the aggregate computational power, with the desired amount of processing power on demand.

The server of the present invention loads tasks onto a single provider computer based in such a way as to match computer capability with task requirements. As tasks are completed, and through the communications channel between the provider computer and the server, additional tasks are fed to the provider computer for execution. All new tasks are placed in a task execution queue on the provider's computer for subsequent execution. To provide maximal control of the execution of tasks on a provider's computer, the CE supports the concept of task priorities and task preemption (tasks can preempt executing tasks of lower priority). These features allow the server to more effectively utilize available provider computer resources.

In order for the server of the present invention to correctly characterize the provider's computer, the server also collects usage and performance histories (profiles) from the provider computer concerning a wide variety of parameters. For example, the server collects Internet connectivity patterns and processor availability patterns during the course of any 24-hour period, weekly period, monthly period, etc. The processing power of the processor itself is also important since certain clients may request higher or lower processing power depending on relative cost and timeliness tradeoffs.

To fulfill its function, the task scheduler must fulfill the following requirements: The task scheduler gets jobs and associated tasks from clients. It must then take those tasks and distribute them to various provider computers for execution. It must also record the fact that it has assigned a particular task to a particular provider computer.

The task scheduler passes both interim and final results to the client and records the amount of work performed to an on-line ledger. It then credits a provider's account with the amount of time that the provider computer actually performed on the specific task. This allows the provider to be paid by the computational work that it has contributed to a particular job. (By definition, time multiplied by the power of the provider computer equals work performed.)

provider computer and the bandwidth that is available to provide elements and tasks to provider computers.

The CTS collects statistics regarding provider computer capabilities. This is critical for both assigning payment to a provider and a debit notice to the client. It is also important that the CTS know at all times the capability of the provider computers.

As noted above, the CTS may send the same tasks to more than one provider computer. This helps the CTS be assured that the computations that are performed by the assigned provider computers are correct. Since results from provider computers should be the same for the same task (unless randomization is integral to the calculation of the task), any differences in response are noted by the CTS and resolved. This assures the client that answers are accurate.

The CTS takes explicit measures to protect the intellectual property inherent in a task. To accomplish this, the CTS obfuscates executable elements and data elements to remove contextual clues to the code's intended purpose. Furthermore, the CTS also removes from tasks all traces of client identity. Thus, unless specifically intended otherwise by the CTS, a provider has no means of knowing the clients for whom elements are executed nor their domain of inquiry.

To characterize and assess the capabilities of provider computers, in order to perform task scheduling effectively, the CTS executes certain benchmark tasks on provider computers. Also, various test tasks with known results are sent to the provider computer to ensure that results returned by the provider computer are accurate. Unexpected results from a provider can be used to identify attempts to submit fraudulent results or otherwise tamper with the normal execution of the system of the present invention.

In operation, the present invention allows the client to specify job and task parameters via the client interface. Using the client interface, the Client selects desired attributes of nodes in a virtual cluster. These attributes include, but are not limited to, CPU speed (in GF), memory, available disk space, intermittent or constant connectivity, bandwidth (in Kbps), number of nodes, and reliability (as measured by consistency of results). The client also

$$\text{Price per hour for computation} = 4.5 + 0.296P^{1.155}$$

where 4.5, 0.296 and 1.155 are configurable constants.

The client is charged an additional amount per GB for data transferred through each of the four communications paths, **150**, **152**, **154**, and **156** of the present invention as illustrated in **Figure 7**. The communications paths **150**, **152**, and **154** represent duplicated data and are a function of the degree of reliability the client achieves. Of the dashed arrows, only one will be executed, and so the client will be charged for only one of these paths.

The client is also charged for data transfer during the course of executing the particular tasks of the client. For example, let d_i be the quantity (in GB) of data transferred from the client to the server. Let d_o be the quantity of data transferred from the server to the client. Given that some of the data will be duplicated for processing by virtual nodes comprised of more than one provider computer, the client's total data transfer charge is given by the following:

$$\text{Total Cost of Data Transfer} = g\left(d_i + d_o\right)\left(\frac{\ln\left(1-r\right)}{\ln\left(1-\pi\right)}+1\right)$$

where g is the transfer cost per leg per GB, r is the requested node reliability, and π is the average provider computer reliability.

Requested node reliability r can be defined for a virtual node of CPU speed C and bandwidth B as the probability of the virtual node completing a task in no more time than the time it would take a fully idle, constantly connected computer with CPU speed C and bandwidth B to complete the task.

Average provider computer reliability π is defined as the probability of the provider's frequency of contact not diminishing, in the near term, from the provider's historic average.

The client is also charged an hourly premium for requesting reliability greater than a set baseline reliability. The charge per hour is:

$$\text{Expected Time-to-Completion} = T_E = \frac{w}{Ca_c} + \frac{d_i + d_o}{Ba_b}$$

As noted earlier the client specifies the CPU speed, C_v, bandwidth, B_v, and reliability, r, of a virtual node. If reliability is not specified, the system of the present sets a default value for the baseline reliability R. The client may select a greater, but not lesser, reliability.

The system then computes the implied "requested" time-to-completion for the task as:

$$\text{Requested Time-to-Completion} = T_R = \frac{w}{C_v} + \frac{d_i + d_o}{B_v}$$

The subset of provider computers that satisfy the following criteria are considered "candidate provider computers":

- The provider computer is not working on a paying job.
- When the client specifies w, d_i, and d_o, the provider computer has (approximately) $T_E = T_R$.
- When the client does not specify w, d_i, and d_o, the provider computer has (approximately) $a_cC = C_v$ and $a_bB = B_v$.

Taking provider computers' CPU's and CPU availabilities as fixed, candidate provider computers have expected bandwidths (bandwidth multiplied by bandwidth availability) that satisfy the following equation:

$$Ba_b = Ca_c \frac{(d_i + d_o)C_v B_v}{wB_v(Ca_c - C_v) + (d_i + d_o)Ca_c C_v}$$

For each candidate provider computer, the system calculates the weighted average availability, π, as:

computer's power rating. Note that, effectively, the provider is being paid for time, not work. However, because the provider computer's power rating is a function of the provider computer's availability, the rate of payment the provider receives is a function of the average work per unit time the provider computer completes over time.

Providers are paid a fixed amount per unit time. The system established a target annual payment per MHz-hour, k, and a baseline reliability, R. A provider computer with a weighted average availability of π and CPU of C is paid $\frac{kC\ln(1-\pi)}{\ln(1-R)}$ per hour. This assumes that the provider computer is working over the whole interval from launch to conclusion of the task.

Provider Payments and Task Scheduling (Full Version)

As part of establishing a given provider computer as a viable source of computing power for the system, for each provider computer, the system constructs a CPU signature and a bandwidth signature. These signatures show the probabilities of the provider computer's CPU and bandwidth being available for a fixed time interval over time. For example:

Beginning of Time Interval	Pr(CPU available)	Pr(Bandwidth available)
1	30%	0%
2	40%	10%
3	50%	20%
4	40%	30%

The number of intervals should be enough to cover a reasonable circadian "cycle" (e.g. 1 week). The signature should be updated periodically.

For each provider computer, the system measures CPU speed by sending to the provider computer certain benchmark tasks that have known response times. The bandwidth is also measured. As noted earlier, these measures are periodically updated.

described, multiple instances of each such module may, in fact, be present in the architecture in order to allow for scalability to larger operations, to increase reliability so that no single point of failure exists in the system, or to have multiple instances of the entire server located at different physical locations to allow for redundancy and insurance against failure.

The server comprises a registration servlet **50**, which accepts registration requests over the network **28** from clients and providers who wish to participate on the system. Registration information is sent by the registration servlet to the registration manager **52** for processing.

The registration manager **52** receives appropriate information from the registration servlet **50** and assigns the client/provider computer to a database. The registration manager then generates a certificate and returns the certificate in the response to the client/provider computer. The provider computer certificate includes the provider public key, the provider-id (which is a hash of the provider public key), and the provider database ID. The client certificate comprises the client/user public key, which then acts as the client/user ID, the client ID, and the provider computer database ID.

The provider servlet **56** accepts various messages from providers as will be discussed more fully below, and routes them to the appropriate provider manager **58**. Further, the provider servlet **56** returns responses from the provider manager to the various provider over the network **28**. The web server of the present invention encrypts and decrypts requests, authenticates requests, routes them to the appropriate servlet, encrypts and returns responses. The provider servlet accepts GET and POST messages from providers. (These messages will be more fully explained below).

Provider manager **58** manages the interaction of the system of the present invention with the various providers. The provider manager receives and routes requests from other internal server components such as the client manager **68**, the task scheduler **64**, and the database server **54**. All administrative information and processing that relates to providers is managed by the provider manager **58**.

The element servlet **60** is used as the point where data elements and code are exchanged between clients and providers over the network. Elements are executables, or

in the database **54**. Various algorithms present in the task scheduler ensure that appropriate tasks are assigned to the appropriate providers given priority of tasking and other prioritizing data.

Database server **54** stores information on clients and providers. To accomplish this task, the database server **54** comprises an account database relating to providers and revenue accumulated by providers, a client database relating to the identity and contact information for clients, job and task information, and a provider database relating to providers and identification and contact information for providers. Such information, while shown as being located on a single database server **54** can also be located on multiple servers to so that the system can be scaled up to handle many more users as well as ensure privacy and security of the information.

The web manager **70** intermediates access between the account section of the web site and the database. The web manager **70** supports the web based registration, update of registration and account information, and retrieval of account-specific and system-wide statistics.

Monitoring and control module **72** comprises a set of utilities that support the overall management of the system of the present invention. The monitoring and control functionality **72** provides graphical user interfaces for system operators to monitor the health of the overall system. The monitoring and control functionality **72** polls all servers for health information in the form of name-value pairs that contains status information and related attributes to all of the servers. In this fashion, the monitoring and control **72** can keep track of the health of the entire system. The monitoring and control module can also respond to asynchronous notification of system events from servers.

Log server **74** contributes to the health of the overall system by collecting debug and error log data. Any errors that are encountered throughout the system whether they are in clients, providers, or any of the servers of the system are collected in the log server for subsequent analysis and debugging where necessary.

As noted above, while single instances of the various modules have been indicated in **Figure 5**, this architecture is not meant as a limitation. Additional instances of any given

accomplished in order to even the workflow and ensure the most efficient functioning of the system of the present invention.

In a similar fashion, web manager **70** can be augmented by additional instances of web managers with requests being allocated among the various web managers. The monitoring and control module **72** can also be represented by multiple instances of the monitoring and control function. There can also be multiple instances of the log server **74**. However, there must be close coordination among the various log servers so that any trends and errors can be detected and corrected.

Registration servlet **50** can also be augmented by additional components. Clients and providers are assigned across the various instances of registration servlets.

The task scheduler **64** maintains an in-memory model of the state of all of the providers (i.e. the provider profiles) and any tasks that it manages. Information to establish the task scheduler is initially retrieved from database **54**. It is thereafter updated as messages are received from the provider manager regarding status of efforts of the various providers, and from the client manger, which forwards relevant messages from clients to the task scheduler.

The task scheduler also requests additional information about providers by cuing a get task message (for retrieving task status) or a get cache contents for obtaining the cache contents of the server for the provider. The task scheduler comprises various scheduling algorithms. This task scheduler makes initial assignments of tasks to providers and migrates tasks from one provider to another if necessary. When the task scheduler schedules a new task, it records the assignment of the task and the provider to which it is assigned in the task database **54**. It further queues a task message to the provider. When migrating a task, the task scheduler removes the task from the current provider by cuing a remove task message, assigning the partially completed task to a new provider via a task message. All such operations are reported in the task database.

The element servlet is the entity that stores data elements that are to be assigned to providers for processing. Clients upload their data elements to the element servlet **60**. The element servlet authenticates the client as one that is permitted to store elements in the file

only access the server if they have a legitimate certificate. In fact, the server uses a hash of the client's public key as a client identifier.

The protocol is reliable because it incorporates sequence numbers, acknowledgement, and retransmission. Combined, these protect against dropped messages, out-of-order processing of messages, and the processing of duplicate messages.

It is worth noting, the low level protocol provides the abstraction of a bi-directional communication channel between components even though it is implemented on top of HTTP that has a request-response model. A bi-directional model is simulated by having the client periodically "poll" the server for messages. This poll is done on an exponential decay to limit the amount of traffic generated by polling. If the CTS does not respond to a particular GET message, perhaps because of heavy CTS load, subsequent GET messages are sent ever more infrequently, according to an exponential decay, until some limit periodicity is achieved. In this manner, heavy CTS can be automatically mitigated through less frequent CE GET requests.

The maximum polling rate is present each time the client and server exchange a high level message. The two basic operations in the low level protocol are the "GET" and the "POST", both of which are initiated by the client. The POST operation sends a new message (or set of messages) to the server. The GET is a poll operation to get any message (or set of messages) that the server may have enqueued for the client. Since the low-level protocol is based on HTTP it takes advantage of that protocol's features for multipart messages and different message encodings (e.g., for compression). Further, multipart messages can be sent in order to maximize communication efficiency. All messages are also subject to data compression to limit the volume of data being transmitted and to conserve bandwidth.

The low-level protocol also supports the concept of session. Sessions are implemented using cookies and provide for the resetting of a communication channel in the event that synchronization is lost.

The following is a listing of high level message types used by the present invention in its communications protocol. This list is illustrative in nature. Other message types can be added to the present invention as the need arises. As noted above, a series of messages are

Request that a DataElement message be sent describing a particular data element

RemoveExecutableElement:

Request the removal of the named executable element.

RemoveDataElement:

Request the removal of the named data element.

Attach:

Attach to a particular task or job. The attach message is used to subscribe to receive status updates on the identified task or job.

GetTask:

Request that the identified task be checkpointed. The checkpoint is returned to the client in a Task message.

GetTaskStatus:

Requests the status of a particular task. The server will send a TaskStatus message in response.

RemoveTask:

Requests the removal of the named task.

ExecutableElement:

Register an executable element with the server. The actual executable element data must be downloaded over the data channel.

DataElement:

Register a data element with the server. The actual data element data must be downloaded over the data channel.

CloseSession:

Close current session.

Error:

Report an error condition.

ExternalDelivery:

Alert the recipient that a message is waiting at a specified URL.

Provider-to-Server messages

NodeStatus:

Reports a change in the status of the provider node.

NodeProfile:

Reports profile information for node (e.g., machine type, available disk, available memory, connection speed, operating system type).

GetConfig:

Request that the server send a Config message to set the providers server-controlled configuration settings.

CacheContents:

Reports the contents of the providers element cache. This information is useful in task scheduling.

Contents:

Reports the contents of the task queue.

Task Status:

Reports the status of a task. There can be both intermediate and final task statuses. Intermediate task statuses may contain partial results. Final task statuses contain the results of the task. Task statuses are also used to report errors in the execution of a task.

Task: The provider sends the task message out in response to a GetTask message from the provider. In this case, the Task message is a checkpoint of the state of the identified task.

Error:

Report an error condition.

ExternalDelivery:

Alert the recipient that a message is waiting at a specified URL.

Heartbeat:

The present invention can also be employed in other manners, such as a method of marketing computers by offering incentives to computer customers that agree to operate a compute engine (CE) on the computers and having the CE utilize idle computational processing power on the computers. Incentives can include, but are not limited to free computer use, free ISP service, discounted computer sales price, discount computer lease price, a sales rebate, periodic rebates, and usage fees. The CE can also be utilized to deliver "pushed" content, such as advertising, to these computer customers via a display window of said computer's graphic interface or via said computer's sound output.

A system and method for the establishment and utilization of networked idle computational processing power has been illustrated. It will be appreciated by those skilled in the art that other variations in, for example, the calculation of power, and the methods for compensating providers can be accomplished without departing from the scope of the invention as disclosed.

8. The system for using excess computational power of claim 5 wherein the run time interface further comprises instructions for attaching the client computer of a job in progress and for checking on the status of a job while the job is in progress.

9. The system for using excess computational power of claim 1 wherein the CTS further comprises instructions for charging the at least one client for computational power used during the execution of the programs.

10. The system for using excess computational power of claim 9 wherein the charging of the client is made when the task is launched.

11. The system for using excess computational power of claim 10 wherein a task is launched when it is scheduled by the CTS to be run by at least one provider computer.

12. The system for using excess computational power of claim 1 wherein the CTS further comprises instructions for monitoring a virtual node specified by the client for failure.

13. The system for using excess computational power of claim 9 wherein charging of clients is based upon the sum of the power of the provider computers running the client tasks.

14. The system for using excess computational power of claim 9 wherein the CTS further comprises instructions for making payments to providers based upon the available computational power the provider's computer.

15. The system for using excess computational power of claim 14 wherein the available computational power is a function of CPU availability, CPU speed and bandwidth availability.

16. The system for using excess computational power of claim 15 wherein CPU speed is measured by running benchmark tests of the CPU periodically.

17. The system for using excess computational power of claim 1 wherein each of the plurality of provider computer further comprises a general purpose compute engine (CE) for executing programs sent from the central server.

18. The system for using excess computational power of claim 17 wherein the CE further comprises instructions for allowing an owner of the provider computer to set the availability parameters of the provider computer.

25. The system for using excess computational power of claim 24 wherein the client is also charged an hourly premium for requesting reliability greater than a set baseline reliability given by the formula:

$$\text{Price per hour for premium reliability} = kCN\left(\frac{\ln(1-r)}{\ln(1-R)}-1\right)$$

where k is a baseline price per MHz-hour, C is the CPU speed requested of a virtual node requested by the client, N is the number of virtual nodes running, r is the requested reliability, and R is the baseline reliability.

26. The system for using excess computational power of claim 1 wherein the owners of the provider computers are paid a fixed amount per unit work for providing excess computational power according to the following formula:

$\frac{T\ln(1-\pi)}{\ln(1-r)}$ per GHz-hour, where

T is a rate, π is an average availability, and r is a target reliability.

27. A method for using computer excess computational power comprising:

adapting at least one client computer to create programs for execution;

connecting the client computer to a network;

connecting a central task server (CTS) to the network, wherein said CTS is adapted to receive the programs from the client;

connecting a plurality of provider computers for providing excess computational power to the network and adapting said provider computers to receive the programs from the central server, the plurality of provider computers providing computational power upon demand.

28. The method for using excess computational power of claim 27 further comprising supplying the at least one client computer with a client application programming interface (API) for allowing the client computer to specify the parameters under which the programs are to be run.

40. The method for using excess computational power of claim 35, further comprising providing the CTS with instructions for making payments to providers based upon the available computational power the provider's computer.

41. The method for using excess computational power of claim 40 wherein the available computational power is determined as a function of CPU availability, CPU speed and bandwidth availability.

42. The method for using excess computational power of claim 41 wherein CPU speed is measured by running benchmark tests of the CPU periodically.

43. The method for using excess computational power of claim 27, further comprising providing each of the plurality of provider computer with a general purpose compute engine (CE) for executing programs sent from the central server.

44. The method for using excess computational power of claim 43, further comprising providing the CE with instructions for allowing an owner of the provider computer to set the availability parameters of the provider computer.

45. The method for using excess computational power of claim 44 wherein the availability parameters comprise the amount of RAM available for the programs, the amount of disk space available for the programs and the times during the day, if any, that the provider computer is available to execute the programs.

46. The method for using excess computational power of claim 44, further comprising providing the CE with instructions for opening a widow on the provider computer when the programs are being run.

47. The method for using excess computational power of claim 44, further comprising providing the CE with instructions for displaying content from the central server in the window.

48. The method for using excess computational power of claim 44, further comprising providing the CE with instructions for executing the programs only when excess computational power is detected on the provider computer.

T is a rate, π is an average availability, and r is a target reliability.

53. A method for the establishment and utilization of networked idle computational processing power comprising:

loading a client API on to client computers to allow client jobs to be broken into independent tasks.

loading compute engine (CE) software onto provider computers to execute tasks using idle computational processing power of said provider computers and using a centralized task server (CTS) that is at least intermittently networked to said client computers at any time selected by said clients and intermittently connected to provider computers to exchange tasks and results.

54. The method of claim 53, wherein said CE runs unobtrusively and securely on the provider computer.

55. The method of claim 53, wherein a task scheduler in said CTS matches computational power requested by clients to provider resources.

56. The method of claim 53, wherein CE operates the provider computer to periodically contact the CTS to retrieve tasks and control messages and/or send results.

57. The method of claim 53, wherein the client API periodically contacts the CTS to retrieve results.

58. The method of claim 53, wherein the CE displays a graphic window when running tasks to display processing and status information.

59. The method of claim 58, further comprising display of advertising on other targeted messages by the CE graphic window.

60. The method of claim 58, wherein the window can be made to disappear based on configurable mouse coordinates.

61. A system for the established and utilization of networked idle computational processing power comprising:

at least one client computer with a client API that allows client jobs to be broken into independent tasks;

charging said clients a fee; and

compensating said providers.

68. The method of creating and intermediating a market for idle computational processing power of claim 67 wherein API software on client computers provides tasks and compute engine (CE) software on said provider computers executes said tasks and provides said results.

69. The system of claim 61, further comprising a communication protocol for governing communication between the API, CTS, and CE, said protocol having separate message and data channels; wherein the message channel is a bi-directional channel between clients and the server that is used to pass high-level messages that facilitate distributed processing and the data channel is a channel used to pass large blocks of data elements and executable elements.

70. The system of claim 61, wherein the client API supports jobs of arbitrary size by being programmed to monitor only task results.

71. The system of claim 61, further comprising a runtime API which defines how tasks interact with their environment.

72. The system of claim 61, wherein the client API contains an interface for task checkpointing.

73. The system of claim 61, wherein the CE supports the queuing of tasks, task prioritization, and preemption of tasks based on priority.

74. The system of claim 61, wherein the CTS and CE together record computer CPU and communications line signatures.

75. The system of claim 74, wherein the signatures are historical usage patterns for use in task scheduling.

76. The system of claim 61, further comprising system-wide secure communications, client anonymity, and obfuscation of client code.

77. The system of claim 61, further comprising means for pricing jobs and scheduling tasks based on delivering clients a virtual cluster comprised of virtual nodes, said

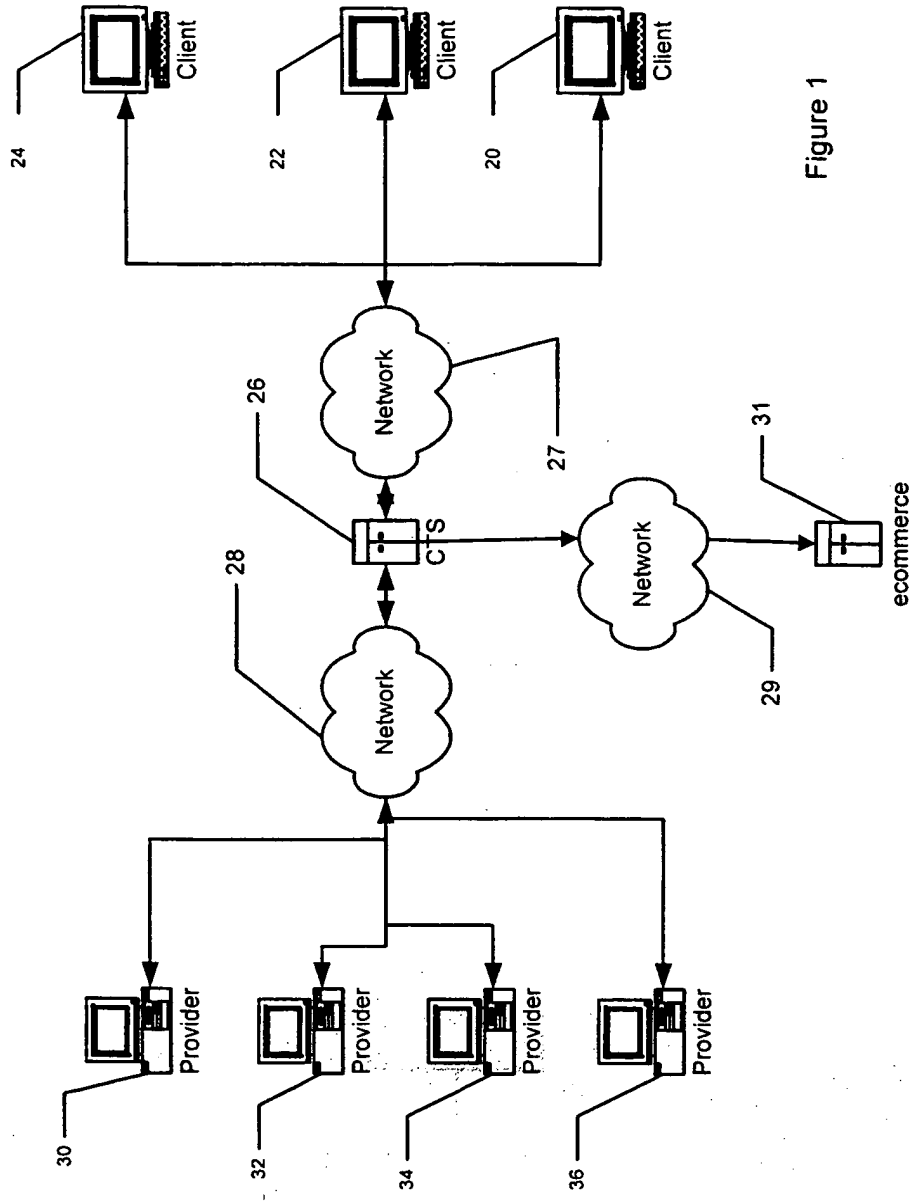

Client
Client
Client
24
22
20
Network
26
27
CTS
28
Network
Network
31
ecommerce
29
Provider
Provider
Provider
Provider
30
32
34
36

Figure 1

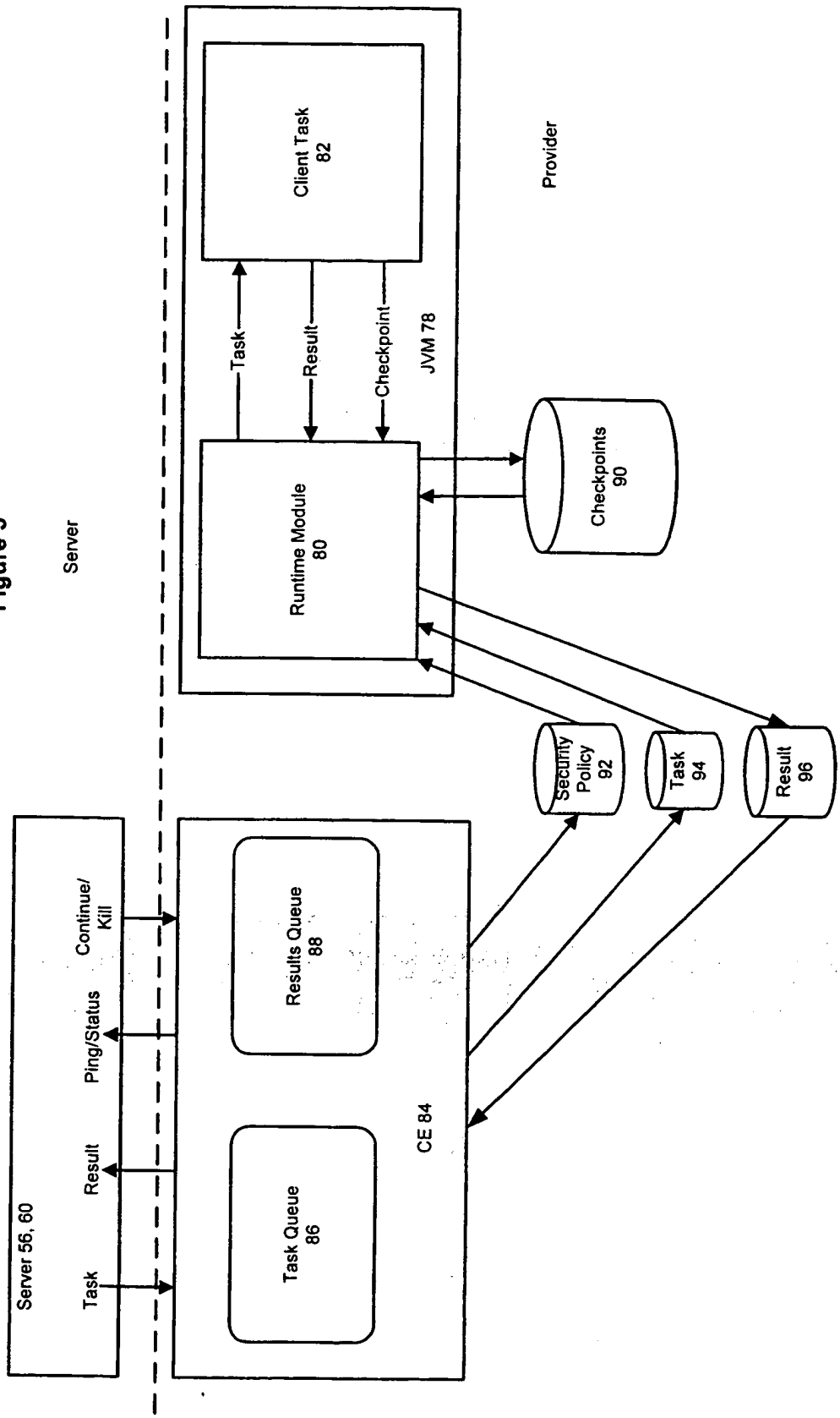

Server
Provider
Client Task 82
Task
Result
Checkpoint
JVM 78
Runtime Module 80
Checkpoints 90
Security Policy 92
Task 94
Result 96
Server 56, 60
Continue/ Kill
Ping/Status
Result
Task
Results Queue 88
Task Queue 86
CE 84

Figure 3

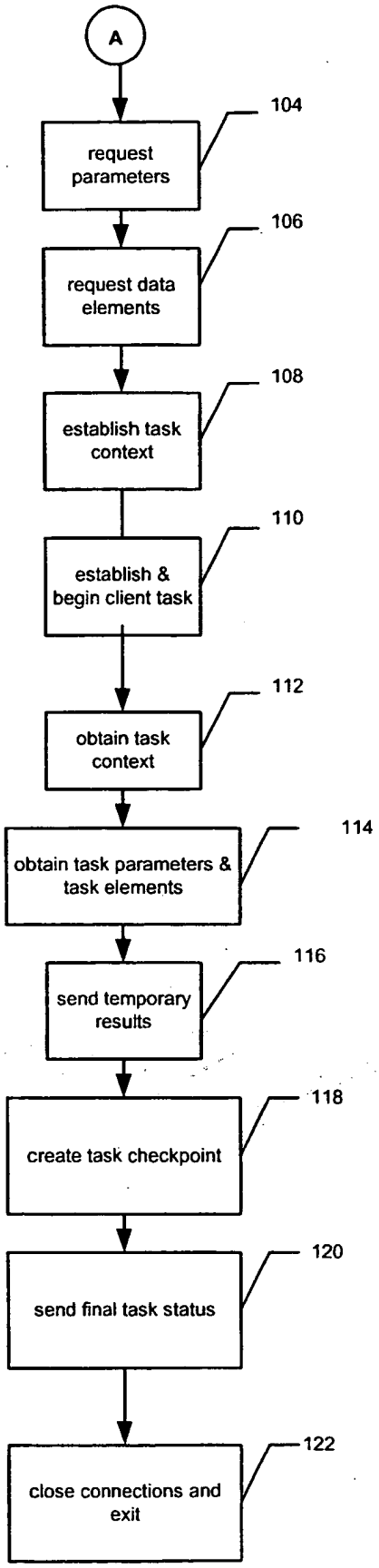
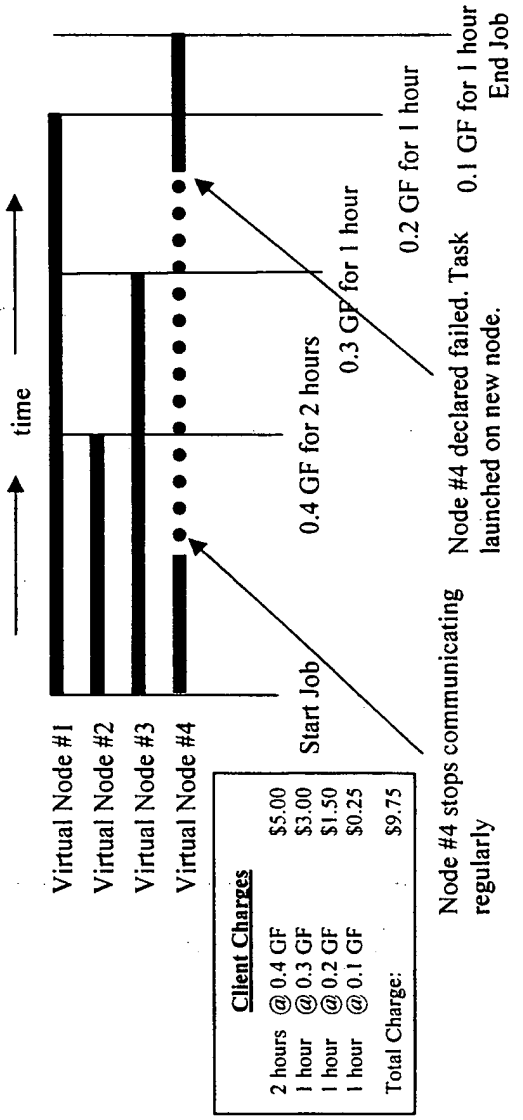

A
request parameters
104
request data elements
106
establish task context
108
establish & begin client task
110
obtain task context
112
obtain task parameters & task elements
114
send temporary results
116
create task checkpoint
118
send final task status
120
close connections and exit
122

Figure 4B

Figure 6


time
Virtual Node #1
Virtual Node #2
Virtual Node #3
Virtual Node #4
Start Job
0.4 GF for 2 hours
0.3 GF for 1 hour
0.2 GF for 1 hour
0.1 GF for 1 hour
End Job
Node #4 stops communicating regularly
Node #4 declared failed. Task launched on new node.
Client Charges
2 hours @ 0.4 GF $5.00
1 hour @ 0.3 GF $3.00
1 hour @ 0.2 GF $1.50
1 hour @ 0.1 GF $0.25
Total Charge: $9.75

(12) INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
1 March 2001 (01.03.2001)

PCT

(10) International Publication Number
WO 01/14961 A3

(51) International Patent Classification[7]: G06F 9/50

(21) International Application Number: PCT/US00/24336

(22) International Filing Date: 28 August 2000 (28.08.2000)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:
60/150,766 26 August 1999 (26.08.1999) US
60/210,334 12 June 2000 (12.06.2000) US

(71) Applicant: **PARABON COMPUTATION** [US/US]; Suite 1000, 3930 Walnut Street, Fairfax, VA 22030 (US).

(72) Inventors: **ARMENTROUT, Steven, L.**; 11879 St. Trinians Ct., Reston. VA 20191 (US). **O'CONNOR, James**; 11625 Ayreshire Road, Oakton, VA 22124 (US). **GANNON, James**; 3817 Inverness Road, Fairfax, VA 22030 (US). **SLETTEN, Brian**; Apt. 906, 900 N. Stuart Street, Arlington, VA 22203 (US). **CIER, Sean**; Apt. 12, 11800 Federalist Way, Fairfax, VA 22030 (US). **CARLSON, Sarah**; 13236 Memory Lane, Fairfax, VA 22033 (US). **DAVIS, Jonathan**; Apt. 312, 202 Columbia Road NW, Washington, DC 20009 (US). **DUPERTUIS, Greg**; 7888 Leroux Lane, Manassas, VA 20112 (US). **MCCLOUGHLIN, Scott**; 2827 28th Street, #14, Washington, DC 20001 (US). **DAVIES, Antony**; 11733 Stuart Mill Road, Oakton, VA 22124 (US).

(74) Agents: **ROBERTS, Jon, L.** et al.; Roberts Abokhair & Mardula, LLC, Suite 1000, 11800 Sunrise Valley Drive, Reston, VA 20191 (US).

(81) Designated States *(national)*: AE, AG, AL, AM, AT, AU, AZ, BA, BB, BG, BR, BY, BZ, CA, CH, CN, CR, CU, CZ, DE, DK, DM, DZ, EE, ES, FI, GB, GD, GE, GH, GM, HR, HU, ID, IL, IN, IS, JP, KE, KG, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, MA, MD, MG, MK, MN, MW, MX, MZ, NO, NZ, PL, PT, RO, RU, SD, SE, SG, SI, SK, SL, TJ, TM, TR, TT, TZ, UA, UG, UZ, VN, YU, ZA, ZW.

(84) Designated States *(regional)*: ARIPO patent (GH, GM, KE, LS, MW, MZ, SD, SL, SZ, TZ, UG, ZW), Eurasian patent (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European patent (AT, BE, CH, CY, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE), OAPI patent (BF, BJ, CF, CG, CI, CM, GA, GN, GW, ML, MR, NE, SN, TD, TG).

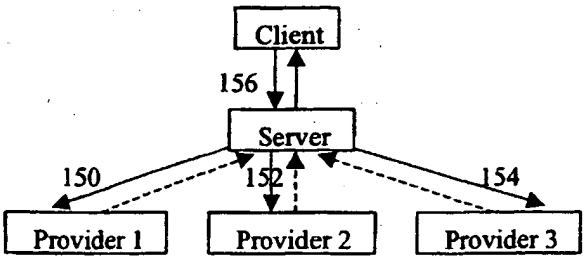
Published:
— *with international search report*

(88) Date of publication of the international search report:
2 August 2001

For two-letter codes and other abbreviations, refer to the "Guidance Notes on Codes and Abbreviations" appearing at the beginning of each regular issue of the PCT Gazette.

(54) Title: SYSTEM AND METHOD FOR THE ESTABLISHMENT AND UTILIZATION OF NETWORKED IDLE COMPUTATIONAL PROCESSING POWER

WO 01/14961 A3



(57) Abstract: A distributed computing platform using the idle computational processing power of a plurality of provider computers is disclosed. At least one networked server collects tasks from client computers, schedules and distributes the tasks to networked provider computers, and collects and returns results to client computers. A client API forms tasks and collects results. A compute engine operates on the provider computers to communicate with the server and execute tasks using idle computational power.

C.(Continuation) DOCUMENTS CONSIDERED TO BE RELEVANT

Category °	Citation of document, with indication,where appropriate, of the relevant passages	Relevant to claim No.
X	CHAO-JU HOU ET AL: "IMPLEMENTATION OF DECENTRALIZED LOAD SHARING IN NETWORKED WORKSTATIONS USING THE CONDOR PACKAGE" JOURNAL OF PARALLEL AND DISTRIBUTED COMPUTING,US,ACADEMIC PRESS, DULUTH, MN, vol. 40, no. 2, 1 February 1997 (1997-02-01), pages 173-184, XP000682281 ISSN: 0743-7315 page 174, left-hand column, line 35 -page 175, right-hand column, last line	1-3,17, 27-29, 43,63, 79,80
A	WALDSPURGER C A ET AL: "Spawn: a distributed computational economy" IEEE TRANSACTIONS ON SOFTWARE ENGINEERING,US,IEEE INC. NEW YORK, vol. 18, no. 2, February 1992 (1992-02), pages 103-117, XP002124500 ISSN: 0098-5589 the whole document	1-81
A	MUTKA M W: "ESTIMATING CAPACITY FOR SHARING IN A PRIVATELY OWNED WORKSTATION ENVIRONMENT" IEEE TRANSACTIONS ON SOFTWARE ENGINEERING,US,IEEE INC. NEW YORK, vol. 18, no. 4, 1 April 1992 (1992-04-01), pages 319-328, XP000328663 ISSN: 0098-5589 the whole document	67,74
A	TANDIARY F ET AL: "Batrun: utilizing idle workstations for large scale computing" IEEE PARALLEL AND DISTRIBUTED TECHNOLOGY: SYSTEMS AND APPLICATIONS,XX,XX, vol. 4, no. 2, 21 June 1996 (1996-06-21), pages 41-48, XP002124512 ISSN: 1063-6552 the whole document	1,17,27, 43,53, 54,56,61